INFIX POSTFIX CLEMENTE PAREDES (VERSION 2.0)

1.Utilisation:

Pour faire la conversion, on doit exécuter au terminal la commande suivante, faisant appel au code c déjà compilé et celui en caml. On doit aussi être situé dans le dossier INFIXPOSTFIX_clemparedesMP2I dans le terminal. (cd Downloads/InfixPostfix_clemparedesMP2I)

$ echo STRING| ./InfixPostfixclem | ocaml InfixPostfixclem.ml

ou STRING doit être remplacé par l'expression en notation infixe, sans espaces, avec des ''. *(Pour éviter des erreurs bash)*

EXEMPLE: $ echo '(2+3)*(4+5)' | ./InfixPostfixclem | ocaml InfixPostfixclem.ml

```
[clementeparedes@fedora INFIX POSTFIX]$ echo "(2+3)*(4+5)"
 | ./InfixPostfixclem | ocaml InfixPostfixclem.ml
45
```

Pour en tester chacun des codes individuellement, la même démarche est faite mais en effaçant le *pipe* à ne pas exécuter.

EXEMPLE: "$ echo '(2+3)*(4+5)' | ./InfixPostfixclem" teste la conversion infix->postfix en C.

2.Choix d'implémentation:

2.1 C

La conversion de notation infixe à Postfix a besoin de l'emploie d'une pile, ou Stack, qui, à travers l'algorithme shunting yard, suit une série de pas pour en arriver à la notation polonaise inverse. Puisqu'on travaille avec un string, on doit avoir une pile de chars, qui va garder quelques éléments à partir de chaque caractère du string.

L'algorithme shunting yard s'emploie comme suit:
(via https://aquarchitect.github.io/swift-algorithm-club/Shunting%20Yard/)

1. For all the input tokens:
    1. Read the next token
    2. If token is an operator (x)
        1. While there is an operator (y) at the top of the operators stack and either (x) is left-associative and its precedence is less or equal to that of (y), or (x) is right-associative and its precedence is less than (y)

        1. Pop (y) from the stack
        2. Add (y) output buffer
    2. Push (x) on the stack
3. Else if token is left parenthesis, then push it on the stack
4. Else if token is a right parenthesis
    1. Until the top token (from the stack) is left parenthesis, pop from the stack to the output buffer
    2. Also pop the left parenthesis but don't include it in the output buffer
5. Else add token to output buffer

2. Pop any remaining operator tokens from the stack to the output

On suit cet ordre d'opérations pour le code dans la fonction order, l.86.

On a aussi symcheck et opcheck, qui vérifient qu'un caractère soit un symbole (opérateurs et parenthèses), ou un opérateur, respectivement. La fonction priorité est aussi employée, pour attribuer à chaque opérateur un entier naturel de 0 à 5 selon son degré de précédence.

Pour pouvoir travailler avec des entiers a plus d'un chiffre, des virgules sont mises entre chaque nombre et opérateur, lesquelles seront traitées par le code en OCaml.

2.1.1 Exemples: Faites à partir de la démarche montrée en [1]

INPUT	OUTPUT
"(2+3)^(5!)"	"2,3,+,!,^"
"23+6-(5^6)"	"23,6,5,6,^,-,+"
"(2+3)/0"	"2,3,+,0,/"

Tous les opérateurs sont bien convertis, et les nombres de plus d'un chiffre sont aussi respectés.

2.1.2 LIMITATIONS:

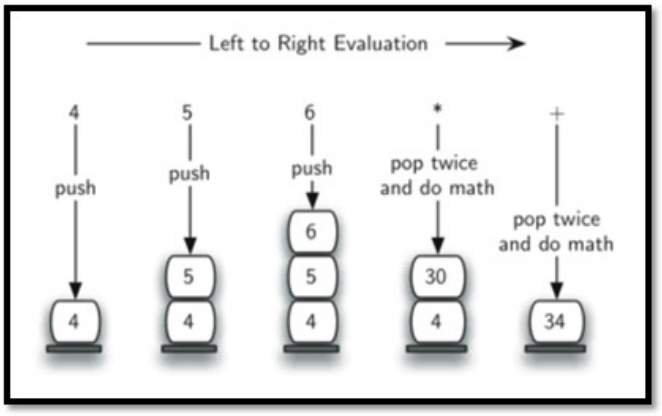
L'input ne doit pas avoir d'espace. (ex: '1 + 2' renvoie '1')

Le résultat peut être négatif, mais le premier élément ne doit pas l'être. (ex: "-2+1" renvoie erreur, "1-2" marche bien.)

L'expression doit avoir que des entiers. (Flottants non implémentés)

2.2 OCaml

En Caml, une pile est aussi utilisée pour évaluer la notation Postfix.



Le string est séparé dans une liste de sous-string en utilisant les virgules comme séparateurs (non-incluses dans les éléments de la liste), et ensuite chaque élément est évalué avec du pattern matching, qui, à partir de la nature du string, suit l'algorithme de l'évaluation postfix (l.20). Comme on a une liste de strings, on doit les convertir en int avec int_of_string, faire le calcul nécessaire, et ensuite les re-convertir en string avec string_of_int.

2.2.1 Exemples: Faites à partir de la démarche montrée en 1

INPUT	OUTPUT
"23,2,+"	25
"25,2,3,+,!"	12025
"2,2,3,4,5,-,/,+,*"	-2

3. Exemples Globales: Faites à partir de la démarche montrée en 1.

INPUT	OUTPUT
(2+3)^(5!)	3426856788313115681
(22/7)-314	-311
1+2*3-4^5/6+2!	-161
2+(6*7)^2	1766
(MP2I+2)-(345*7)	-2411
1/0	Exception: Division_by_zero

(L'entrée accepte des lettres, mais elles ne sont pas considérées au moment du calcul)